Towards an Adaptive Middleware for Efficient Multi-Cloud Data Storage

Ansar Rafique, Dimitri Van Landuyt, Vincent Reniers, Wouter Joosen

imec-DistriNet, KU Leuven
3001 Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract

A multi-cloud storage architecture combines different storage technologies and resources from multiple clouds. As it allows application providers to manage the risks associated to technology or vendor lock-in, provider reliability, data security, privacy, it is an increasingly popular tactic for designing the storage tier of cloud-based applications.

Multi-cloud storage architectures are however prone to run-time dynamicity: many dynamic properties impact the way such a setup is governed and evolved over time, e.g., storage providers enter or leave the market; pricing policies, QoS metrics and SLA guarantees change over time; etc.

This paper provides an in-depth discussion of this complexity, and sketches our architectural vision of self-adaptive middleware solutions that monitor and change the storage architecture (semi-)autonomously. We highlight two areas of ongoing and future research that deserve specific attention: (i) systematically monitoring the storage systems, despite heterogeneity, and (ii) providing uniform methods and techniques to change and control the different storage resources dynamically.

Keywords Adaptive data management middleware, Efficient multi-cloud data storage, Multi-Tenant SaaS, NoSQL

1. Introduction

Cloud storage is an important service of cloud computing, which offers storage-as-a-service, supports different database technologies (both SQL and NoSQL), and allows data owners to store their data in the cloud [18]. Cloud data storage frees data owners from the burden of maintaining an expensive on-premise storage infrastructure and offers economies of scale benefits [16], delivered through elastically scalable, pay-per-use schemes.

Despite all these benefits, cloud data storage also raises concerns over provider reliability, data security, performance, and availability [6, 15]. In addition, relying on a single cloud storage provider (CSP) increases the risk of technology or vendor lock-in and places limits on the application and service-level-agreement (SLA) requirements that can be provided. Therefore, multi-cloud storage architectures, which combine storage resources and database technologies from multiple cloud providers is becoming increasingly popular [2, 6, 10, 11].

Data placement decisions across a multi-cloud architecture are commonly based on the static properties of the operational environment. These multi-cloud data placement decisions do not take into account the behavior of individual storage systems and dynamically changing conditions of cloud providers and their related storage systems. Relevant dynamic properties are: performance characteristics (i.e. latency and throughput), free space size, evolving price conditions, new providers arrival, cloud provider availability (i.e. uptime), etc. Hence, not taking such properties into account may lead to sub-optimal data placement decisions in highly dynamic environments. Therefore, we argue in favor of *adaptive middleware* platforms that monitor and enact upon dynamically changing conditions, and are capable of performing multi-objective optimization (e.g., in terms of performance, storage price, cloud availability, data security etc.) at a fine-grained level (i.e. for different customers).

In this paper, we present an initial blueprint of an *adaptive middleware* platform that (i) provides continuous monitoring capabilities and takes into account run-time metrics (i.e. dynamic properties), (ii) adapts constantly to changing conditions for making run-time decisions, (iii) optimally decides the cloud provider well suited for the data storage taking numerous multi-objective optimization factors (e.g., performance, pricing policy, disk storage space, peak-load condition, etc.) into consideration, and (iv) autonomously decides and enacts changes to the environment (i.e. when observ-

CrossCloud'17, April 23, 2017, Belgrade, Serbia.

http://dx.doi.org/10.1145/3069383.3069387

ing ongoing performance or scalability issues, autonomously scaling-up some storage nodes). As such, this architecture is structured according to the well-known MAPE-k architecture for self-adaptive systems [8, 9].

The remainder of this paper is organized as follows: Section 2 motivates the need for adaptive middleware solutions. Section 3 describes our envisioned middleware architecture. Section 4 then provides an in-depth discussion of the implementation feasibility and highlights the main directions for future research. Section 5 reviews the related work, whereas Section 6 concludes the paper.

2. Motivation

The motivation for this paper is based on our frequent interaction and experiences with industry-level Software-as-a-Service (SaaS) providers [14]. The illustrative example for this paper –a multi-tenant document processing application– is introduced in Section 2.1. Section 2.2 subsequently highlights three scenarios that require active monitoring capabilities of changing conditions for making suitable data placement decisions.

2.1 Document Processing SaaS Application

The multi-tenant document processing SaaS application is a B2B cloud offering that generates and archives batches of documents (e.g., invoices, payslips, etc.) for a wide group of customer organizations (i.e. tenants). The tenants of this SaaS application vary widely, e.g., banks, hospitals, telecom operators, etc., and each tenant imposes different non-functional requirements when it comes to performance, scalability, and availability, usually expressed in SLAs. As an example, telecom operators produce a large set of monthly bills, usually at the end of each month. For these tenants, high availability and elastic scalability of storage resources is particularly relevant during seasonal peaks (e.g., at the end of each month). To address these requirements, the document processing SaaS application uses a multi-cloud storage architecture and combines on-premise storage resources (i.e. limited in terms of resources) with external public cloud providers (i.e. supporting elastic scalability and higher availability for seasonal spill-over).

2.2 Scenarios

However, in reality, managing a multi-cloud storage architecture manually is a tedious task because such an architecture involves heterogeneous resources from multiple clouds, which exert properties that may evolve quickly. For example, cloud providers may update their pricing policies or change their Quality-of-Service (QoS) guarantees. In consequence, managing such an architecture requires continuous attention, careful planning, and appropriate manual intervention. The complexity of managing a multi-cloud storage architecture for multi-tenant SaaS applications makes such task even more difficult as different storage requirements and SLAs for different tenants must also be taken into account.

Table 1 illustrates a number of scenarios and the required interventions to realize them. The remainder of the section elaborates further on three key scenarios that require continuous monitoring of changing run-time conditions of storage resources in a multi-cloud environment.

Scenario #1: Performance Optimization. Dynamic data placement decisions will entail selecting the most suited cloud storage provider by taking into account different performance characteristics of storage systems (static property), different performance profiles (dynamic property based on measurement and performance profiling/predictions), and also the differences between tenant SLAs. For example, in the document processing SaaS application, more strict performance guarantees are required for banks, as opposed to supermarkets. The first row of Table 1 illustrates the type of dynamically changing storage policy that accommodates such environmental changes.

Scenario #2: Peak-load Conditions. Although all tenants of the document processing SaaS application have the same functional requirements, they usually have very different non-functional requirements with respect to performance and availability. However, the requirement with respect to sending out the invoices, all tenants have strict deadlines and invoices are usually sent at the same time (i.e. end of each month). To accommodate such peak periods where a large number of users connect simultaneously to the SaaS application, an adaptation of the storage architecture involves dynamically and temporarily including spill-over resources (see #2 in Table 1 for possible adaptation actions).

Scenario #3: Evolving Pricing Schemes. Pricing schemes differ across cloud providers and evolve over time. Consider a scenario where a new cloud provider enters the market that offers more cost-efficient data storage or temporary discounts. As the cost optimization is one of the core objectives of multi-tenant SaaS applications, there is a clear incentive to dynamically extend the storage architecture by incorporating the new provider and maximally utilizing its storage resources (see #3 in Table 1 for possible adaptation actions).

3. Middleware Architecture

Dynamically adapting the system behavior requires an architecture that provides active monitoring capabilities and supports (re)configuration at run-time. Figure 1 presents the overall architecture which consists of four layers. From top to bottom they are: (i) the *Multi-tenancy* layer, (ii) the *SaaS Application* layer, (iii) the *Adaptive Data Management* layer, and (iv) the *Multi-Cloud Storage Architecture* layer.

The *Multi-tenancy* and *SaaS Application* layers provide tenant-specific and application-wide configuration and customization (e.g., back-end configurations and SLAs specifications). The *Multi-Cloud Storage Architecture* layer provides a uniform API which underneath consists of a number of database-specific drivers for different back-end storage systems operating at different cloud storage providers.

Table 1: An overview of multi-cloud scenarios and their expected adaptation actions required to accomplish these scenarios.

#	Scenarios/Conditions	Adaptation Actions
1	CSP2 outperforms CSP1 (i.e. performance optimization)	1. Change storage policy to use CSP2 for the future data storage requests (instead of CSP1) 1.1. Keep existing data in CSP1 1.2. Migrate existing data from CSP1 to CSP2
2	CSP2 is suffering from ongoing performance issues in peak-load condition	1. Add more storage nodes in CSP2 (i.e. scale-out) 2. Temporary spill-over to CSP1
3	CSP2 offers a discount and storage price drops below that of CSP1 (i.e. cost optimization)	1. Change storage policy to use CSP2 for the future data storage requests (instead of CSP1) 1.1. Keep existing data in CSP1 1.2. Migrate existing data from CSP1 to CSP2
4	The SLA of CSP1 offers higher availability than that of CSP2	1. Use CSP1 for the data, which requires higher availability 1.1. Keep existing data in CSP2 1.2. Migrate existing data from CSP2 to CSP1

However, the core of the middleware and the focus of this paper is the *Adaptive Data Management* layer (positioned in the center of Figure 1), which we describe in detail in the rest of this section. We mainly focus on the roles and responsibilities of different components of the *Adaptive Data Management* layer and how they efficiently and flexibly support three key scenarios discussed in Section 2.2.

Adaptive Data Management

The *Adaptive Data Management* layer provides adaptation capabilities for responding to changes at run-time and meeting with different SLAs requirements specified by each tenant. The layer is comprised of five main components: (i) the *Data Access* component, (ii) the *SLA Management* component, (iii) the *Multi-Objective Decision* component, (iv) the *Monitoring* component, and (v) the *Reconfiguration Support* component as shown in Figure 1. This section further discusses the responsibilities and capabilities of each component of the *Adaptive Data Management* layer with respect to three scenarios discussed in this paper. However, due to space constraints, this paper focuses on scenarios #1 and #2.

Data Access Component. The dynamic multi-cloud data placement decisions must take into account the requirements of the individual tenant for a certain data type. In a simplistic case (e.g., storing invoices for tenants), multiple data stores can be elected as suitable candidates for data placement decisions. However, the question now raises which properties must be considered to efficiently select the most optimal data stores. In order to make this decision efficiently, the current state and dynamic properties of cloud storage providers and their associated storage systems must be taken into consideration. Therefore, in such a scenario, to perform an efficient data placement decision, the *Data Access* component gets the persistence configuration details of different storage systems, distributed across multiple cloud providers from the `Persistence Management Component` (step 1 in Figure 1) and the SLA requirements specified by the tenant from the *SLA Management* component (step 2 in Figure 1) and passes the information to the *Multi-Objective Decision* component (step 3 in Figure 1). The latter component is responsible for making appropriate optimization decisions. The *Data Access* component, then makes data placement operations (step 5 in Figure 1), based on the returned information from the *Multi-Objective Decision* component about the optimal and most suited cloud storage providers.

SLA Management Component. The *SLA Management* component stores SLA requirements specified by tenants. As different tenants have different requirements, the component exposes an interface that allows tenants to specify SLAs, which vary significantly among them and usually are expressed in terms of different optimization objectives (e.g., performance, cost, and availability).

Multi-Objective Decision Component. The dynamic data placement decisions are influenced by many run-time factors. For example, SLA requirements, dynamic conditions of a multi-cloud storage environment (e.g., performance, availability, cost, etc.), and individual requirements for a certain data type may influence the decision. In addition, as different tenants have different requirements, the *Multi-Objective Decision* component must make optimal data placement decisions based on these run-time factors and also take appropriate optimization decisions for different tenants. To ensure that, as part of step 4 in Figure 1, the *Multi-Objective Decision* component sends a request to the `Storage Monitor` sub component of the *Monitoring* component, which continuously monitors different QoS metrics (see Listing 2 as an example of monitored QoS metrics for the write performance) and sends the response back (i.e. monitored metrics) to the *Multi-Objective Decision* component. The *Multi-Objective Decision* component compares the monitored metrics (i.e. the QoS) with the expected performance SLAs specified by the SaaS application or their tenants (see Listing 1 for an example of expected SLA policy for the write performance). For example, as shown in Listing 2, three data

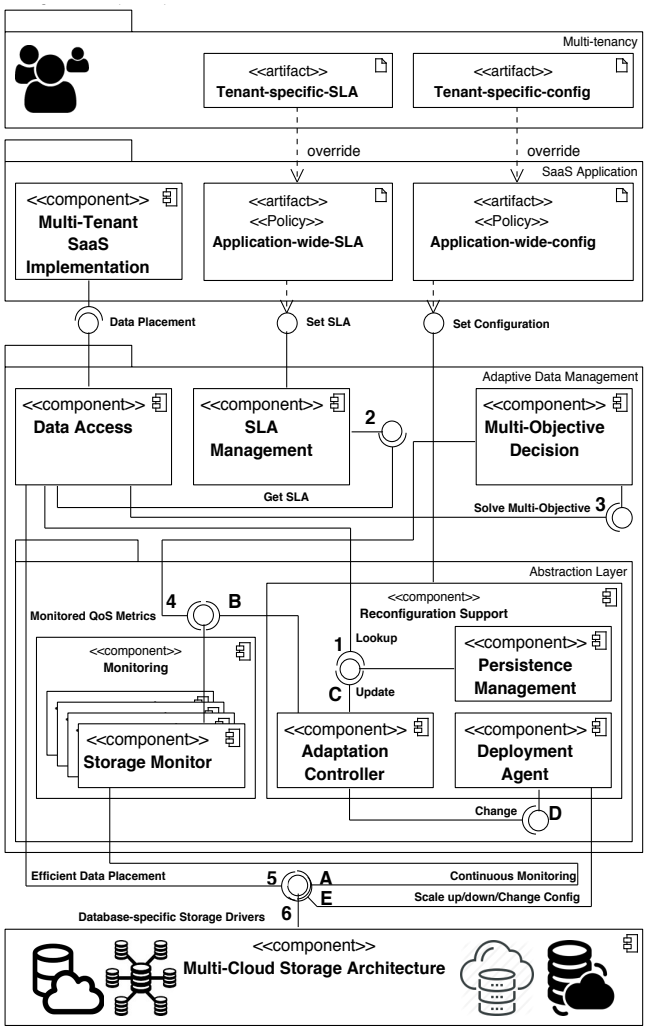

Figure 1: High-level overview of the envisioned architecture.

stores on potentially varying cloud providers (i.e. Cassandra-Private, Cassandra-Public, and MongoDB-Private) satisfy the imposed SLA requirements for the invoices generated from bank (see Listing 1). The *Multi-Objective Decision* component based on the SLA requirements and monitored QoS metrics makes optimization decisions as such selects the most efficient cloud provider suited for data storage.

Listing 1: An example of expected write performance SLA policy for tenants (e.g. bank and telecom operator).

Data Type	Bank	Telecom Operator
Invoices	< 4ms	< 6ms
Payslips	< 5ms	< 8ms
...	...	...

Monitoring Component. The *Monitoring* component is responsible for monitoring different SLA metrics (i.e. the QoS metrics) of back-end storage systems operating at different cloud providers. This component exposes an interface to retrieve the monitored QoS metrics information. The `Storage Monitor` component continuously monitors QoS metrics (step A in Figure 1) and stores up-to-date QoS metrics in the database. The up-to-date monitored QoS metrics are accessed by different components of the *Adaptive Data Management* layer for different purposes. For example, to perform efficient multi-cloud data placement decisions, the up-to-date QoS metrics are accessed by the *Multi-Objective Decision* component. Similarly, to autonomously (re)configure a multi-cloud storage architecture or to be able to react to unusual demand situations (e.g., add more nodes, change the replication factor, etc.), the monitored QoS metrics information is also accessed by the `Adaptation Controller` sub component of the *Reconfiguration Support* component. An example of monitored QoS metrics (i.e. latency and free disk space size) is shown in Listing 2.

Listing 2: Monitored QoS metrics for the write performance of storage systems operating at different cloud providers.

Provider	Invoice	Payslip	Space
Cassandra−Private	2ms	2ms	40%
Cassandra−Public	3ms	4ms	70%
Mongodb−Private	3ms	3ms	50%
Mongodb−Public	5ms	6ms	80%
...	...	...	...

Reconfiguration Support Component. The *Reconfiguration Support* component provides an interface to set configuration details and performs an initial deployment and configuration of heterogeneous storage systems distributed across multiple clouds. The component is comprised of three sub components (i) the `Persistence Management` component, (ii) the `Adaptation Controller` component, and (iii) the `Deployment Agent Service` component.

The `Persistence Management` component contains the persistence configuration details of different storage systems and for that, the component provides an interface to lookup and update these details. The `Adaptation Controller` component is responsible for the management of resources and triggers an appropriate action (e.g., install new instances to a database cluster, change the replication factor, etc.) if the resources are suffering from various ongoing issues or if the unusual demand situations have occurred. The component reads the up-to-date monitored QoS metrics (step B in Figure 1) from the `Storage Monitor` sub component of the *Monitoring Component*, which provides continuous monitoring capabilities. The `Adaptation Controller` component contains a number of (re)configuration rules and based on these rules and the monitored QoS metrics, takes an appropriate action. For example, as shown in Listing 3, the (re)configuration rule to keep the average latency of the Elasticsearch below 80 ms. The appropriate (re)configuration action (i.e. adding a new storage node), specified in the (re)configuration action executes in case of service violation. To perform such an action, as part of step D in Figure 1, the component sends a change signal to the `Deployment Agent` component, which is responsible for providing the needed deployment support.

Listing 3: The (re)configuration rule specifies the requirement to keep the average latency below 80 ms. Add more nodes in case of service violation.

```
rule "Keep the average latency below 80ms"
  when
  sla: SlaMeasurement(storage == "Elasticsearch"
    and latency > 80)
  then
  sla.setExtraNodeRequired(true)
end
```

The `Deployment Agent` component is responsible for making the desired (re)configuration and deployment (step E in Figure 1) including: adding more nodes in a cluster (i.e. scale up), remove nodes from the cluster (i.e. scale down), changing consistency level and replication factor, etc.

4. Implementation Feasibility

The blueprint architecture presented above represents the overarching vision of a number of ongoing implementation and analysis efforts that are each aimed at overcoming specific challenges of multi-cloud storage architectures.

A first set of challenges relates to harmonizing data placement decisions in light of heterogeneity of different storage resources, different database technologies and their respective APIs, and the use of policies to externalize the data distribution logic from the application (depicted in Figure 1). Taking these aspects into account, we are studying existing abstraction layers and data access platforms [12] (such as Impetus Kundera, Apache Gora, Data Nucleus Access Platform, etc.), as well as existing multi-cloud APIs (such as Apache jclouds, CloudRail, etc.), and also ongoing standardization efforts (such as CDMI, CIMI, etc.). For expressing and enforcing storage policies and rules that influence data placement decisions, we are looking at rule engines such as JBoss Drools, Open Rules, JRuleEngine, etc.

Monitoring a multi-cloud storage architecture in practice is highly challenging, first and foremost because of the heterogeneity between databases (NoSQL databases) and database APIs (REST, JPA). In addition, cloud storage providers commonly only provide data access to their offering (e.g., via a REST API) without providing any further insight into the internal state. Finally, different monitoring components are required to monitor different metrics (e.g., performance, availability, price etc.). For example, to monitor the performance QoS metrics, white-box performance monitoring techniques cannot always be trusted nor always provided [13] and we are particularly looking at how black-box metrics may be indicative of the service health parameters of interest. We envision an extensible architecture in which different `Storage Monitor` components can be plugged in to monitor different QoS metrics (as depicted in Figure 1). Again, to address the complexity and accomplish the challenges involved in monitoring a multi-cloud storage architecture, abstraction mechanisms are required that provide a uniform API to monitor different storage systems and services, offered by different cloud providers.

The process of expansion and contraction of resources at run-time to cope with ongoing performance and scalability issues or to address contradicting requirements of tenants requires elasticity. Although NoSQL databases are designed to be elastic, they are however not autonomously elastic as in an on-premise center an external component is required to take an action when necessary (e.g., adding or removing nodes). Therefore, similar to the *Monitoring* component, the *Reconfiguration Support* component is also faced with heterogeneity in different NoSQL databases, their APIs, deployment models, and the way (re)configuration is performed for each NoSQL database. For example, to perform dynamic (re)configuration for Cassandra and MongoDB databases, the techniques proposed by [5] and [7] could be adapted respectively. We are currently investigating existing configuration management tools such as Docker, Puppet, Ansible, Chef, etc. to implement the desired runtime reconfiguration support.

5. Related Work

Recently, there has been an increasing research interest in multi-cloud storage systems such as MCDB [1], DepSky [2], HAIL [3], ICStore [4], Hybris [6], SPANStore [17] to meet different non-functional requirements of the application. These systems leverage multiple cloud storage providers either to enhance availability, ensure data security, or distribute the trust across clouds. However, these are single-objective optimization solutions in the sense that each proposed system focuses on a specific non-functional requirement. In contrast, our research focuses on multi-tenant SaaS applications in which multi-objective optimization decisions are made for specific tenants. Furthermore, in the context of a multi-tenant SaaS application, where tenants have slightly different non-functional requirements, the prior work mostly focuses on the IaaS layer and thus limited tenants customization support can be provided, whereas our research primarily focuses on the SaaS layer. In addition, none of these multi-cloud storage systems supports flexible data storage policies in function of data management.

In previous work [11], we have presented a middleware that provides fine-grained control over data storage decisions in a multi-cloud storage architecture, and makes data placement decisions that are policy-driven and are based on static properties. As discussed above, such static decision logic leads to sub-optimal data storage decisions when the multi-cloud storage architecture evolves dynamically. The architecture presented in this paper as such extends such a static policy-driven setup with support for policies that are based on dynamic conditions of a multi-cloud storage architecture.

Scalia [10] is a cloud brokerage solution that makes data placement decisions based on data access patterns subject to storage cost optimization. Similarities exist in many as-

pects such as data placement strategy and the use of multiple cloud storage providers to achieve better availability. However, Scalia is a single-purpose solution that mainly focuses on cost optimization, whereas our proposed adaptive middleware is in principle a multi-purpose solution that supports numerous run-time factors such as performance, availability, and performs multi-objective cost optimization.

6. Conclusion

In this paper, we motivated that the dynamic and rapidly evolving conditions should be taken into consideration for making efficient multi-cloud data placement decisions. We advocated a middleware-based solution and presented an initial vision of such a generic and adaptive middleware, which (i) continuously monitors the changing conditions of storage systems operating at different cloud providers, (ii) constantly adapts to the continuous changes in the operating environment and makes efficient multi-cloud data placement decisions, subject to various run-time optimization objectives, and (iii) autonomously identifies the behavior of individual storage systems and changes in the operational environment and finally takes appropriate actions accordingly.

As discussed, this work fits into our ongoing research on application-level middleware for federated data storage architectures.

Acknowledgments

We thank Emad Heydari Beni for his helpful comments and constructive feedback. This research is partially funded by the Research Fund KU Leuven (project GOA/14/003 - ADDIS), the SBO DeCoMAdS project, and the iMinds SeClosed project.

References

[1] M. A. Alzain, B. Soh, and E. Pardede. Mcdb: Using multi-clouds to ensure security in cloud computing. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 784–791, Dec 2011.

[2] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, November.

[3] Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM, 2009.

[4] Christian Cachin, Robert Haas, and Marko Vukolic. Dependable storage in the intercloud. Technical report, Research Report RZ, 3783, 2010.

[5] E. Casalicchio, L. Lundberg, and S. Shirinbab. Energy-aware adaptation in managed cassandra datacenters. In *2016 International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 60–71, Sept 2016.

[6] Dan Dobre, Paolo Viotti, and Marko Vukolić. Hybris: Robust hybrid cloud storage. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 12:1–12:14, New York, NY, USA, 2014. ACM.

[7] M. Ghosh, W. Wang, G. Holla, and I. Gupta. Morphus: Supporting online reconfigurations in sharded nosql systems. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2016.

[8] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[9] Frank D. Macias-Escriva, Rodolfo Haber, Raul del Toro, and Vicente Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267 – 7279, 2013.

[10] Thanasis G. Papaioannou, Nicolas Bonvin, and Karl Aberer. Scalia: An adaptive scheme for efficient multi-cloud storage. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 20:1–20:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[11] Ansar Rafique, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen. Policy-driven data management middleware for multi-cloud storage in multi-tenant saas. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 78–84. IEEE, 2015.

[12] Ansar Rafique, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen. On the performance impact of data access middleware for nosql data stores. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2016.

[13] Arnaud Schoonjans, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen. On the suitability of black-box performance monitoring for sla-driven cloud provisioning scenarios. In *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware*, ARM 2015, pages 6:1–6:6, New York, NY, USA, 2015. ACM.

[14] SeClosed. Secure, cloud-based storage and processing of sensitive documents. http://www.iminds.be/en/projects/SeClosed, 2016. [Last visited on November 23, 2016].

[15] Y. Singh, F. Kandah, and Weiyi Zhang. A secured cost-effective multi-cloud storage in cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 619–624, April 2011.

[16] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 247–258, New York, NY, USA, 2013. ACM.

[17] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V. Madhyastha. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 292–308, New York, NY, USA, 2013. ACM.

[18] K. Yang and X. Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1717–1726, Sept 2013.